Sensor talking to Google Sheets

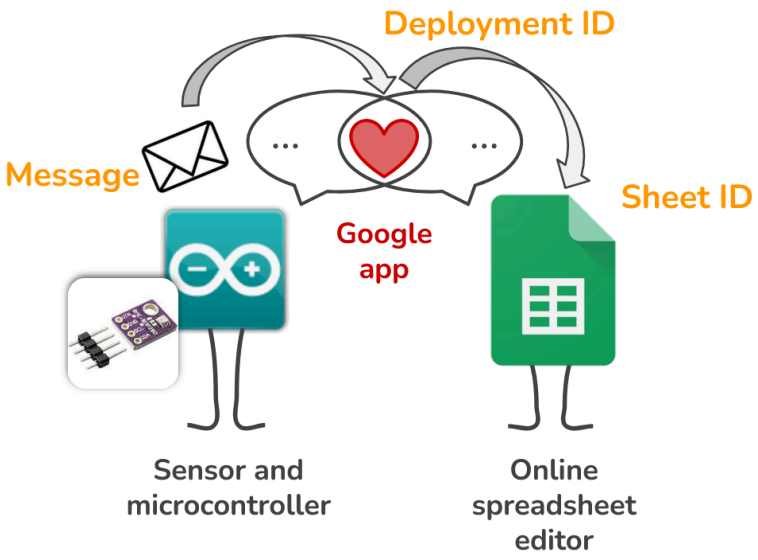
A love story for a talkative sensor

March 23, 2024

By Erika Ronchin

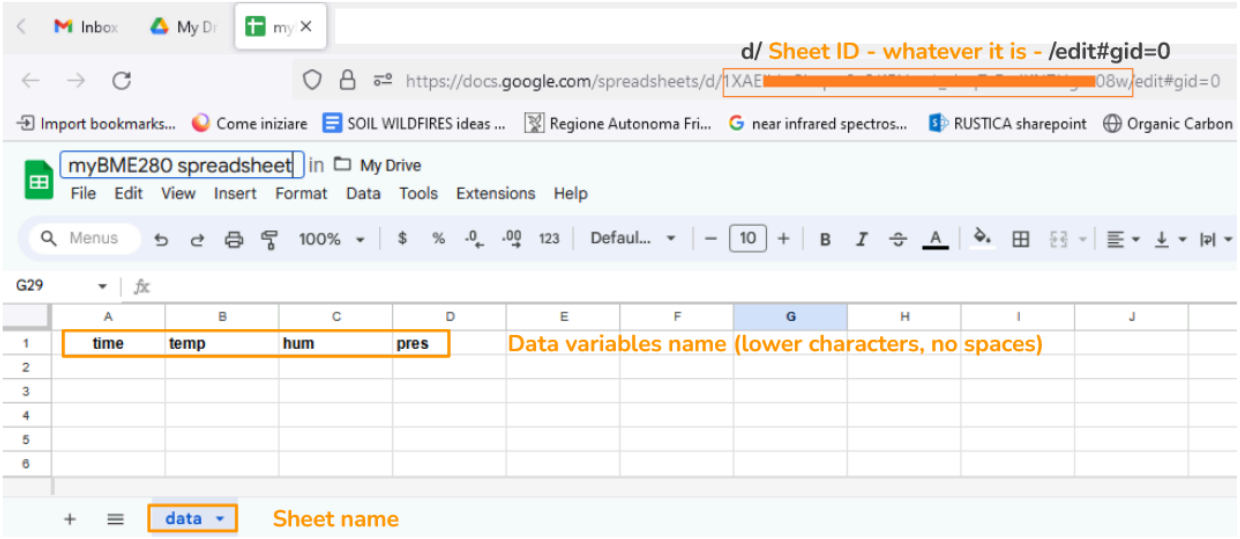
Goal: ESP32 Data Logging to Google Sheets with Google Scripts

Inspired by the awesome explanation by iotdesignopro.com
https://iotdesignpro.com/articles/esp32-data-logging-to-google-sheets-with-google-scripts



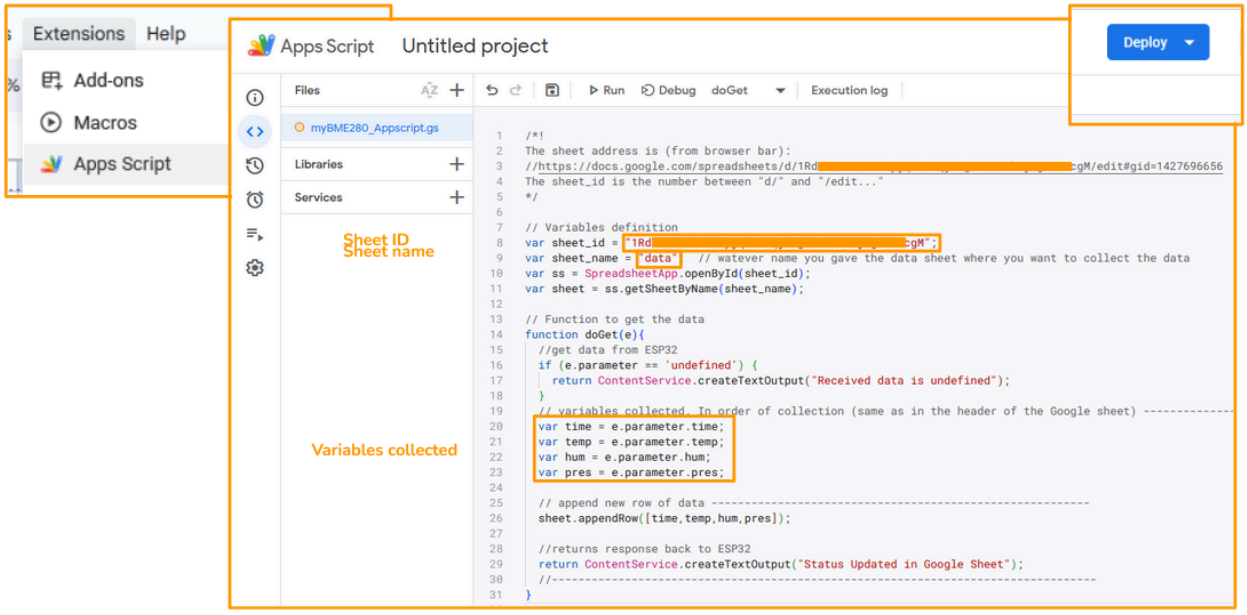
Setup the Google Sheet for Data Logging

Each column will collect one type of data. The column name will be used as pointers to push the data. The column title should be one word and no upper case is allowed. If you want to use multiple words for the title, then add a hyphen in between each word instead of space.



Create a Google app script

The following Google Apps Script is created into **Extensions -> Apps Script**. It is designed to receive data from an ESP32 device via a HTTP GET request and append that data to a Google Sheets document.



This script is mainly built of two parts: variables definition and a function. The app essentially acts as an endpoint for the ESP32 device to send data to. When the ESP32 makes a HTTP GET request to this endpoint with parameters for time, temperature, humidity, and pressure, this script adds that data as a new row to the specified Google Sheets document. It then sends back a response confirming that the data has been successfully added.

1. **Variables Definition**:
    - sheet_id: This variable stores the ID of the Google Sheets document where the data will be stored.
    - sheet_name: This variable stores the name of the sheet within the Google Sheets document where the data will be appended.
    - ss: This variable uses SpreadsheetApp.openById() to open the spreadsheet using the provided ID.

- sheet: This variable uses getSheetByName() to retrieve the sheet within the spreadsheet using the provided name.

2. **doGet(e) Function**:
   - This is a special function in Google Apps Script that gets called when a GET request is made to the URL of the script.
   - It takes an e parameter, which represents the event object containing information about the request.
   - Inside the function:
     - It first checks if the received data is undefined. If so, it returns a message indicating that the received data is undefined.
     - Then, it extracts the parameters time, temp, hum, and pres from the request's parameters (e.parameter), which are sent from the ESP32 device.
     - After that, it appends a new row to the sheet with the received data.
     - Finally, it returns a message indicating that the status has been updated in the Google Sheet.

To send data you will need the APP URL + a string for the data in the following format

https://script.google.com/macros/s/ your deploy ID /exec?**time=300&temp=20&hum=45&pres=101074**

Arduino sending data to a server via HTTP

1. Some libraries, settings, and the registration of your deployment ID

The following parts go BEFORE the void setup(){}

```
#include <WiFi.h>
#include <HTTPClient.h>       // to send data to Google Sheets
#include <Wire.h>             // For the BME280 sensor
#include <BME280I2C.h>        // For the BME280 sensor by Tyler Glen

#define SERIAL_BAUD 115200

// BME280
  const uint8_t sda = 21; // SDA (Serial Data) ->  21 on TTGO-T1, A4 on Uno/Pro-Mini, 20 on Mega2560/Due, 2 Leonardo/Pro-Micro
  const uint8_t scl = 22; // SCK (Serial Clock)  ->  22 on TTGO-T1, A5 on Uno/Pro-Mini, 21 on Mega2560/Due, 3 Leonardo/Pro-Micro
  BME280I2C bme;    // Default : forced mode, standby time = 1000 ms
  float temp(NAN), hum(NAN), pres(NAN);

// Network credentials (STATION)
  const char* ssid = "[redacted]";
  const char* password = "[redacted]";

// Google Deployment script ID and more
  String GOOGLE_SCRIPT_ID = "[redacted]jkH";    // change Deployment
  String urlFinal;  // char for the url with data to log
```

Setting up the variables used in communicating with a Google Sheet via Google Apps Script. The GOOGLE_SCRIPT_ID variable holds the unique identifier of the bound Google Script project and urlFinal is intended to store the final URL for sending data to the Google Sheet but is left uninitialized here.

Here is an explanation of each part of this script:

1. String GOOGLE_SCRIPT_ID = "DEPLOYMENT ID of your Google app"; This line declares a string variable named GOOGLE_SCRIPT_ID and assigns it a value. This value is the unique identifier (ID) of a Google Apps Script project, specifically a script bound to a Google Sheet. This script is

responsible for handling incoming data from the Arduino and updating the Google Sheet accordingly. It's important to note that users need to replace this ID with their own Google Script deployment ID.

2. String urlFinal; This line declares a string variable named urlFinal. This variable is intended to hold the final URL that will be used to send data to the Google Sheet. It's initialized without a value, indicating that it will likely be constructed or assigned a value later in the program, in the void loop(){}

2. The function to send data to a Google Sheet via HTTP

```
39  // FUNCTIONS ------------------------------------------------------------
40    // f9) send data to google sheet
41      void sendtogoogle(String urlFinal){
42          int httpCode;
43          String payload;  // response data received from the server.
44
45          HTTPClient http;
46          http.begin(urlFinal.c_str()); //  Initializes the HTTP client to ma
47          http.setFollowRedirects(HTTPC_STRICT_FOLLOW_REDIRECTS);
48          httpCode = http.GET();  // stores the response code from a HTTP GET
49          Serial.printf("HTTP Status Code: %d\n",httpCode);
50          payload = http.getString();
51          http.end();
52          //http.clear();
53          if (httpCode > 0) {                                        // gett
54            Serial.println("Payload--> "+payload);
55          } else {
56            Serial.println("Not 0 - Payload--> "+payload);
57          }
58      }
59
```

This part can go BEFORE the void setup(){} or at the end of the script

Function written for Arduino. The function sends data to a Google Sheet via HTTP GET request and retrieves the response. It sends an HTTP GET request to a specified URL (the Google Sheet endpoint) and prints the HTTP status code and response payload to the Serial monitor based on the response received.

Here's a breakdown of what each part does:

1. void sendtogoogle(String urlFinal) This function takes a String parameter urlFinal, which contains the URL to which the data is being sent.
2. HTTPClient http; This line initializes an instance of the HTTPClient class, which is used for making HTTP requests.

3. http.begin(urlFinal.c_str()); This line initializes the HTTP client to make a request to the URL specified by urlFinal. The .c_str() function converts the String to a C-style string (const char*), which is the format expected by the begin() function.
4. http.setFollowRedirects(HTTPC_STRICT_FOLLOW_REDIRECTS); This line sets the HTTP client to follow redirects strictly.
5. httpCode = http.GET(); This line executes an HTTP GET request to the URL specified earlier and stores the response code in the variable httpCode.
6. Serial.printf("HTTP Status Code: %d\n",httpCode); This line prints the HTTP status code received from the server.
7. payload = http.getString(); This line retrieves the response body from the server and stores it in the variable payload.
8. http.end(); This line closes the HTTP connection.
9. The if-else block checks if the HTTP response code is greater than 0. If it is, it prints the payload to the Serial monitor. Otherwise, it prints a message indicating that the response code was not greater than 0.

3. The setup(){}

```
62  // SETUP -------------------------------------------------------
63  void setup(){
64    delay(1000);
65    Serial.begin(SERIAL_BAUD);
66
67    // Connect to WIFI and print the accessed WIFI
68    WiFi.begin(ssid, password);
69      Serial.println("\nConnected to the WiFi network");
70      Serial.println("IP Address: " + WiFi.localIP().toString());
71      Serial.print("RSSI: ");
72      Serial.println(WiFi.RSSI());
73
74    // BME280 check connection
75    while(!Serial) {} // Wait
76    Wire.begin(sda, scl);
77    while(!bme.begin())
78    {
79      Serial.println("Could not find BME280 sensor!");
80      delay(1000);
81    }
82  }
```

This part consists of the `setup()` function, which is a standard function in Arduino sketches that is called once when the microcontroller is powered up or reset. In summary, this script initializes serial communication, connects to a WiFi network, and initializes communication with a BME280 sensor over the I2C bus. It provides feedback via serial communication throughout the process. More in detail the script does:

1. **Delay**: The script starts with a delay of 1000 milliseconds (1 second). This delay allows some time for initialization before the serial communication is started.
2. **Serial Communication Initialization**: The `Serial.begin(SERIAL_BAUD)` function initializes serial communication with a specified baud rate. The baud rate is defined by the constant named `SERIAL_BAUD`. This function allows the Arduino to communicate with the computer, over a serial connection.
3. **WiFi Connection**: The script attempts to connect to a WiFi network using the `WiFi.begin(ssid, password)` function. The `ssid` and `password` variables should contain the credentials (SSID and password) of the WiFi network to which the Arduino will connect. After successfully connecting to the WiFi network, it prints information about the connection, including the local IP address and the Received Signal Strength Indication (RSSI).
4. **BME280 Sensor Initialization**: The script initializes communication with a BME280 sensor. It waits for the serial connection to be established (`while(!Serial) {}`). After that, it initializes the I2C communication with the BME280 sensor using the `Wire.begin(sda, scl)` function, where `sda` and `scl` are the pin numbers for the data (SDA) and clock (SCL) lines of the I2C bus, respectively. Finally, it checks if the BME280 sensor is detected by attempting to begin communication with it using the `bme.begin()` function. If the sensor is not found, it prints a message indicating that the sensor could not be found.

4. The script to construct the URL with parameters containing data for Google Sheet

The void loop(){}

```
84  void loop() {
85
86    // 1) Read BME280 sensor
87    BME280::TempUnit tempUnit(BME280::TempUnit_Celsius);
88    BME280::PresUnit presUnit(BME280::PresUnit_Pa);
89    bme.read(pres, temp, hum, tempUnit, presUnit);
90    Serial.println("Humidity (%) = " + String(hum) + " , Temperature (°C) = " + String(temp) + " , Pressure (Pa) = " + String(pres));
91
92    // 2) log data to Google sheet
93    unsigned long seconds = millis() / 1000;
94    urlFinal = "https://script.google.com/macros/s/"+GOOGLE_SCRIPT_ID+"/exec?time="+seconds+"&temp="+temp+"&hum="+hum+"&pres="+pres;
95    Serial.println("POST data to spreadsheet: "+urlFinal);
96    sendtogoogle(urlFinal);
97
98    delay(5000);
99  }
```

This script constructs a URL with parameters containing data such as time, temperature, humidity, and pressure. It then prints the constructed URL to the Serial monitor for verification and sends it to a Google Sheet using the sendtogoogle function.

Here's a breakdown of what each part does:

1. unsigned long seconds = millis() / 1000;: This line calculates the number of seconds since the Arduino board started running using the millis() function. millis() returns the number of milliseconds since the Arduino board started running, and by dividing it by 1000, you get the number of seconds.
2. urlFinal = "https://script.google.com/macros/s/"+GOOGLE_SCRIPT_ID+"/exec?time="+seconds+"&CO2ppm="+CO2ppm+"&temp="+temp+"&hum="+hum+"&pres="+pres;: This line constructs the URL for the Google Sheets endpoint to which the data will be sent. It includes parameters such as time (in seconds since the Arduino started), temperature (temp), humidity (hum), and pressure (pres). These parameters are concatenated to the URL string.
3. Serial.println("POST data to spreadsheet: "+urlFinal);: This line prints out the constructed URL to the Serial monitor for debugging purposes. It allows you to see the exact URL that will be used to send the data to the Google Sheet.
4. sendtogoogle(urlFinal);: This line calls the sendtogoogle function and passes the constructed URL (urlFinal) as an argument. This function is responsible for sending the data to the Google Sheet using an HTTP request, as explained in the script explanation of point 2.